U.S. NAVAL ACADEMY
COMPUTER SCIENCE DEPARTMENT
TECHNICAL REPORT

Low Level Segmentation for Imitation Learning Using the Expectation Maximization Algorithm

Warner, Andrew D.

USNA-CS-TR-2005-04

May 3, 2005

USNA Computer Science Dept. ◦ 572M Holloway Rd Stop 9F ◦ Annapolis, MD 21403

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **03 MAY 2005**	2. REPORT TYPE	3. DATES COVERED **00-05-2005 to 00-05-2005**

4. TITLE AND SUBTITLE **Low Level Segmentation for Imitation Learning Using the Expectation Maximization Algorithm**	5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **U.S. Naval Academy,Computer Science Department,572M Holloway Rd Stop 9F,Annapolis,MD,21403**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**13**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Low Level Segmentation for Imitation Learning Using the Expectation Maximization Algorithm

Midshipman 1/C Andrew D. Warner
United States Naval Academy

1 Introduction

Imagine a robot that is able to develop skills on its own, without being programmed directly. This robot would be invaluable in any business, factory, or laboratory. Unfortunately, this problem, known as inductive learning, is very difficult, and has several varieties. One such is imitation learning. The overall process of imitation learning begins with one robot observing another robot performing a task. The watcher then breaks down, or segments, the demonstrating robot's actions into basic actions called planning units. Next the observing robot uses the planning units to create a plan that accomplishes the required task. The execution of a successful plan demonstrates that the robot has correctly implemented an inductive learning process. The scope of this research does not allow the problem of imitation learning to be discussed in its entirety; however, it does investigate an important subset of the larger problem. This paper focuses on the segmentation of the data, specifically how to break it up into the steps that provide the building blocks of the robots ultimate plan.

2 High Level Problem[1]

Imitation learning is a process by which a robot learns to plan through demonstration and imitation. The goal of inductive learning is to create a plan. In this paper, planning refers to how the robot sequences planning units to accomplish the task at hand. Imitation learning involves two levels, low and high. The high level involves sequencing the planning units to create a plan. In most implementations of imitation learning the robot is preprogrammed with the planning units needed to create the plan. Low level imitation learning focuses on learning the planning units. In low level learning, the planning units are unknown and the robot must first determine what they are in order to create a high level plan. High level learning that ignores low level issues has been well explored [2,3]. However, in a true imitation scenario, a robot must first learn the basic operations required to complete a task before it can learn how to sequence them to solve the problem.

For example, if a machine needed to create a plan to pick up a rock and move it across the room, the machine would need the planning units for moving, picking up, and setting down. The planning unit "pick up" is made of low level actions such as moving the arm, grabbing onto the rock, lifting the rock, and holding the rock. Each planning unit is composed to many low level actions; these actions are what low level imitation learning take into effect.

This paper focuses on how to segment the observed data into the actions that create the planning units. This is the first step in beginning to understand low level issues. If the observing robot is going to imitate the demonstrator's actions, then it needs to understand what those actions are. With our research we attempted to create a method that is able to analyze the observed data and break it up into basic movements or actions that can be used to create the planning units which are ultimately used to create the final plan. This research provides the foundation for implementing low level imitation learning.

3 Proposed Solution

The process of breaking up the observed data into the basic actions is called segmenting. Segmenting is traditionally done by hand. The designer looks at the data and determines where the actions begin and end. It can also be done through heuristics, meeting a predetermined set of conditions which dictate when an action is complete. Another way to segment would be to have the demonstrator send a signal whenever a basic step is complete this provides direct guidance on when to segment. However, this requires the cooperation of the demonstrator and prevents completely independent learning. An additional approach to segmentation is to use the statistical properties of the data. This method is based upon the idea that every basic step is composed of continuous and repetitive actions that change in character when the steps are completed. It appears that points of discontinuity in the data indicate the breakpoints between actions.

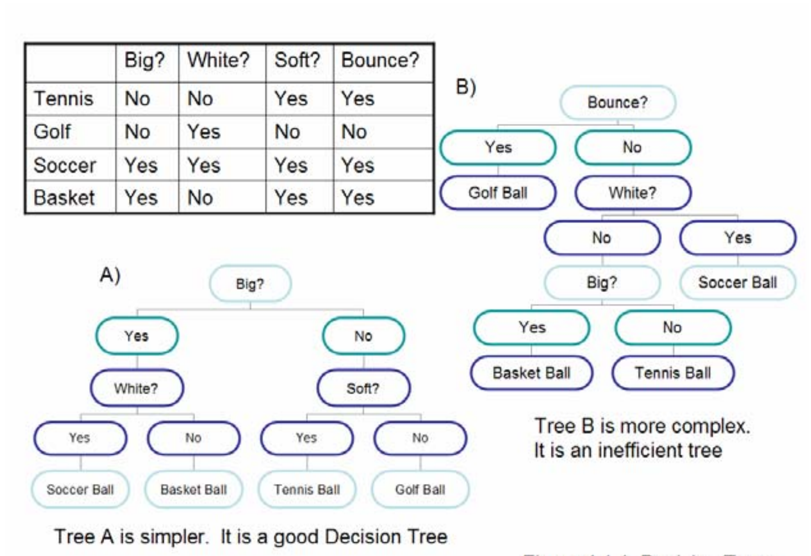
The data collected from the demonstrating robot is in vector form. Each vector contains several numbers that represent the states of the robots motors. We believe that throughout a particular action the states of the motors remain relatively constant. Segmenting the data will break it up into groups with similar properties. Each of these groups will represent a low level action. By segmenting the data, we will know which data points represent a particular action.

This paper discusses the four methods studied to segment the observed data using its statistical properties. Throughout the course of the research we evaluated Decision Trees, Bayesian Networks, Kalman Filters, and the Expectation Maximization algorithm. Each of these methods had different strengths and applications; however, for the project, we chose to implement the Expectation Maximization algorithm (EM).

4 Methods Studied[4]

4.1 Decision Trees

A decision tree outlines the process for making decisions and provides a simple and easy to understand introduction to inductive learning. The basic units of a decision tree are sets of attributes that describe a situation or an object. The simplest version uses Boolean attributes (attributes with a true or false answer). The tree, as shown in figure 4.1.1 is a hierarchical set of questions used to classify data. The tree is created using several different data points; the more data the more accurately the tree can make



	Big?	White?	Soft?	Bounce?
Tennis	No	No	Yes	Yes
Golf	No	Yes	No	No
Soccer	Yes	Yes	Yes	Yes
Basket	Yes	No	Yes	Yes

Figure 4.1.1: Decision Trees

decisions. There are many different ways to create a decision tree and many different trees can be created using the same data. The trick is to make the tree as small and simple as possible by testing the most important attribute first. A simple algorithm, which works well on small data sets, creates decision trees by combining the attributes in every possible order, i.e. forming every possible tree. Note that not every attribute must be used in the creation of the tree. For example, in figure 4.1.1, the attribute Bounce is not necessary in tree A. Next a greedy algorithm goes through and selects the shortest tree. The greedy algorithm simply compares two trees and then keeps the one that is shorter. The algorithm compares every tree and is not efficient, but it is adequate for this example. More efficient algorithms use information theory to select the most useful attributes for the root of the tree. Figure 4.1.1 shows two trees created using the same data. Both trees are correct; however tree A is obviously shorter and simpler then tree B.

Decision trees are designed for data that can be labeled or expressed with propositions. In “How To” manuals and trouble shooting guides the data is labeled, consequently they are often written like a decision tree. However, Decision Trees are impractical for our situation because the data is unlabeled. Our data consist of several numbers in vector form, it is not easily labeled. Decision trees, while not a practical solution to our problem, provide an easily understood introduction to learning algorithms.

4.2 Bayesian Networks

Bayesian Networks are a type of directed graph useful for concisely showing conditional probability distributions. Each node in the graph is conditionally dependent on its parents and independent of its children. In conditional probability, *a* is conditionally dependent on *b* because the existence of *b* alters the likelihood of *a*. This is represented mathematically as $P(a|b) = x$, and means that the probability of *a* given that

$$P(a \mid b) = \frac{P(a \wedge b)}{P(b)} \qquad P(b \mid a) = \frac{P(a \wedge b)}{P(a)}$$

(Conditional Probability)

$$P(a \wedge b) = P(a \mid b)P(b) = P(b \mid a)P(a)$$

$$P(a \mid b) = \frac{P(b \mid a)P(a)}{P(b)}$$

$$P(b \mid a) = \frac{P(a \mid b)P(b)}{P(a)}$$

Figure 4.2.1: Bayes' Rule

only *b* is known is *x*. Bayes' Rule, derived in figure 4.2.1, is the method used to solve conditional probabilities. Bayesian networks provide a visual representation of the probabilities solved by Bayes' Rule.

While Bayesian networks are a concise way of visually representing conditional probability, they are not a practical solution for the segmentation problem. Visually representing the data is not necessary because we are trying to discover the distributions of the data. While knowing the conditional probabilities of the data would be nice, we want to be able to segment the data into basic actions, and it is unlikely that conditional probabilities alone will allow us to do this.

4.3 Kalman Filters

Kalman filters predict the next state of a system based on a linear function with Gaussian noise added. For example, assume that an 8-ball is rolling across a pool table. This motion is the linear function. In a perfect world, the ball would continue on its path and the linear function would always predict the ball's exact position. However, in an imperfect world there is always variance in the ball's motion; other balls, the table rail, or an unleveled table. This variation in the ball's movement is not accurately modeled by a linear function; adding Gaussian noise to the function creates a more accurate model. Kalman filters use conditional probability to change the Gaussian variance to model the actual movement of the ball. If the ball were to bounce off of the table's rail, the model adjusts to fit ball's new motion.

The way that Kalman filters modify the Gaussian is similar to the EM algorithm; however, Kalman filters are used for modifying linear functions. We believe that our data does not reflect a linear function but is better represented by a cluster of data points. Kalman filters are important because of their ability to update the Gaussian variance which provides the foundation for the EM algorithm.

$$\ln(P(x)) = \ln\left(\frac{1}{2\pi^{d/2}\sqrt{|\varepsilon|}} \cdot e^{-1/2(x_j-\mu_i)^{\mathsf{T}}\varepsilon^{-1}(x_j-\mu_i)}\right)$$

$$\ln(P(x)) = \ln\left(\frac{1}{2\pi^{d/2}\sqrt{|\varepsilon|}}\right) \cdot -1/2(x_j-\mu_i)^{\mathsf{T}}\varepsilon^{-1}(x_j-\mu_i)\ln(e)$$

$$\ln(P(x)) = -\ln\left(2\pi^{d/2}\sqrt{|\varepsilon|}\right) - 1/2(x_j-\mu_i)^{\mathsf{T}}\varepsilon^{-1}(x_j-\mu_i)$$

Figure 4.4.1: Simplification using ln()

4.4 Expectation Maximization Algorithm (EM)

The purpose of the EM algorithm is to learn hidden variables. For a variable to be a hidden variable it must not be observable within the available data. Many real world problems have hidden variables in them. For example, if someone has a heart disease, the disease itself is not observable; however, the symptoms and causes are. Using the variable heart disease ties all of the symptoms and causes together and greatly simplifies the conditional probabilities. The EM algorithm outlines a process for discovering values for these hidden variables. Because it is a process, the algorithm can be applied to many different situations and data. For our research we chose to study the EM algorithm as it applies to a mixture of Gaussian distributions. A mixture of Gaussians refers to several different distributions that represent a group of data. Each data point belongs to a specific Gaussian; however, the combined data is accurately modeled by the several Gaussian distributions. Thus, the data is represented by a mixture of Gaussians. The brilliance of the EM algorithm is that, when given a seemingly random set of points generated by several independent Gaussian distributions, it will very nearly recreate the original Gaussian distributions.

The EM algorithm uses an iterative process that is initialized with random numbers. For example, in the case of the mixture of Gaussians, each Gaussian's mean and covariance matrix are initialized with randomly. Each cycle through the algorithm adjusts the means and covariance matrix, bring them closer to the correct values. Every iteration of the EM algorithm is guaranteed to be a more precise estimate because this algorithm maximizes the likelihood of the multi-dimensional Gaussian probability function. Likelihood, in this paper, refers to the log of the probability function. Taking the log of this function enables us to replace multiplication with addition, making the equation easier to work with when taking the derivative. Figure 4.4.1 shows what happens when the log of the multidimensional probability is taken. Maximizing the likelihood guarantees a better estimate after every iteration.

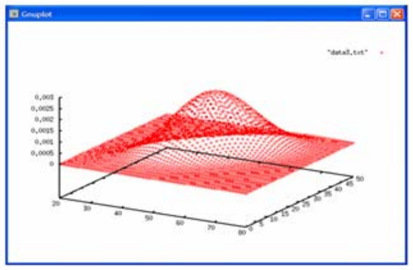
There are two basic steps to EM, the E-step and the M-step. The E-step computes the probability of the current mean and covariance of each Gaussian given the data. The M-step then maximizes the probability that Gaussians generated the data by adjusting the mean and covariance matrices. This process will be explained in greater detail later in this paper.

$$P(x) = \frac{1}{2\pi^{d/2}\sqrt{|\varepsilon|}} \cdot e^{-\frac{1}{2}(x-\mu)^{\mathsf{T}}\varepsilon^{-1}(x-\mu)}$$

$$x = \text{point}\begin{bmatrix} x \\ y \end{bmatrix} \quad \varepsilon = \text{covarience}\begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \mu = \text{mean}\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}$$

$$d = \text{dimensions}$$

Figure 5.0.1: Multivariate Gaussian Distribution



5.0.2: 3-D Plot of a Gaussian Distribution

After studying the methods discussed above we determined that the EM algorithm is the best match for our research. Our data consist of unlabeled vectors that are tied together by Gaussian distributions. As discussed earlier, the Gaussian implementation of EM is designed to handle this type of data. Thus, because the EM algorithm can be applied to our problem we choose to implement it to segment the data. If the EM algorithm works, it will find a set of parameters for a mixture of Gaussian distributions that will define each of the basic steps (we assume that the data points representing each action will cluster into Gaussian distributions). With those distributions the robot will be able to properly segment the data and begin building plans that incorporate the newly learned actions.

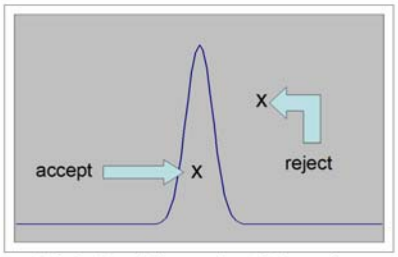
5 Implementation

In order to understand the EM algorithm, we first implemented it on a set of two-dimensional Gaussian distributions with known means and covariance matrices. We began with the simplest multidimensional example in order to understand the algorithm before progressing on to implementing the dimensions of the data.

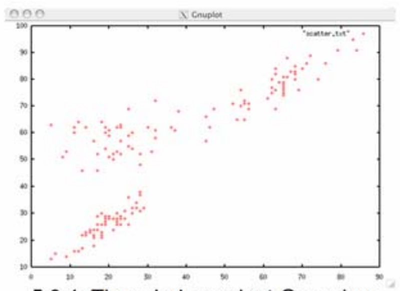
The first step to implement the EM algorithm was to create a valid set of test data. This data was generated using the function in figure 5.0.1. As inputs, this function takes a data matrix (x), a mean matrix (μ), and a covariance matrix (ϵ). The covariance matrix, which describes the shape of the Gaussian function, is symmetrical along the diagonal and its length and width are equal to the number of dimensions of the problem. The mean and data matrices have a variable for every dimension and are stored as one column matrices. The mean matrix indicates where the Gaussian function is centered. Each Gaussian distribution has one mean and covariance matrix that is constant, these matrices defines the distribution. Figure 5.0.1 shows the x, μ, and ϵ for a two-dimensional Gaussian distribution

The test Gaussians were formed by choosing random numbers for the mean and covariance matrices. The resulting plot is shown in figure 5.0.2. The x and y values are from the data points entered into the function. The z value is generated by the Gaussian function; it represents the probability that that point was generated by the Gaussian. The z values give each point its height which gives the graph its hill shape.

Next we created thousands of random points with x, y, and z values. The z values were compared to the answer of x and y being plugged into the Gaussian. If the



5.0.3: One Dimensional Gaussian


5.0.4: Three Independent Gaussians

randomly generated z value was less then or equal to the z values generated by the Gaussian, then the (x, y) point was accepted. This process generates random numbers drawn from the Gaussian distributions; it is illustrated with the one dimensional example in figure 5.0.3. Points below the bell-curve are considered to be part of the Gaussian and are accepted.

After repeating the data generation process three times, to create three independent and random Gaussians, we had created the test data necessary to implement a two dimensional Gaussian version of the EM method correctly. The x and y values of the test data are plotted in the graph in fig 5.0.4, the three major groups represent the three Gaussian distributions. This graph is the raw data.

With the two dimensional data available, we began implementing the EM algorithm. As written earlier, the two basic steps in the EM algorithm are the E and M-step. In the E-step the expected probabilities of the Gaussians based on the data is generated. The M-step then maximizes the mean and variance in order to best represent the data. Because the M-step maximizes the likelihood of the data, every pass through the EM algorithm improves the probability that the new mean and variances were generated by the data. After several passes through, the means and variance represent the data better than the original Gaussians that generated them. This is because the randomly generated data is an incomplete representation of the Gaussian that created it and accurately modeling it requires a different Gaussian than the one that created it. The more points that are available for analysis, the closer the Gaussian generated by the EM algorithm will represent the original mean and covariance.

Bayes' Rule

$$P_{ij} = P(c=i \mid x_j)$$
$$P_{ij} = P(x_j \mid c=i) \cdot P(c=i) \cdot P(x_j)$$

Values Needed to Solve P_{ij}

$$P(c=i) = \omega_i$$
$$P(x_j \mid c=i) = P(x_j) \text{(in Figure 5.0.1)}$$
$$P(x_j) = ?$$

How to Solve $P(x_j)$

$$1 = P(c=1 \mid x_j) + P(c=2 \mid x_j) + P(c=3 \mid x_j)$$
$$1 = \frac{P(x_j \mid c=1) \cdot \omega_1}{P(x_j)} + \frac{P(x_j \mid c=2) \cdot \omega_2}{P(x_j)} + \frac{P(x_j \mid c=3) \cdot \omega_3}{P(x_j)}$$
$$P(x_j) = P(x_j \mid c=1) \cdot \omega_1 + P(x_j \mid c=2) \cdot \omega_2 + P(x_j \mid c=3) \cdot \omega_3$$

$$P_{ij} = \frac{P(x_j \mid c=i)\omega_i}{P(x_j \mid c=1) \cdot \omega_1 + P(x_j \mid c=2) \cdot \omega_2 + P(x_j \mid c=3) \cdot \omega_3}$$

Figure 5.1.1: Using Bayes' Rule to solve for P_{ij}

5.1 E-Step

Simply put, the E-step calculates the probability of the data generating each Gaussian. Figure 5.1.1 uses Bayes Rule to break down the derivation of the E-step, which is explained in the following paragraphs.

The E-step is relatively simple to implement. The first iteration through the algorithm uses randomly generated means, variances, and component weights as starting points. Because we know that the data was generated by three distributions, three Gaussians are used in the E-step (i=3). Knowing how many Gaussians to use in the E-step is one of the trickiest parts of using EM. Using too few results in the exclusion of some of the data and using too many results in several Gaussians covering the same set of points. The only way to figure out how many Gaussian distributions are present is to either plot the data and infer the number (figure 5.0.4 clearly shows three distributions), or to work the algorithm several times and try to deduce when the right number of Gaussians have been discovered.

Once the number of Gaussians has been decided and the initial means, covariance and component weights have been created, it is time to solve for P_{ij}, the probability that the ith Gaussian was created by the jth data point, x_j. This process begins with Bayes' Rule. Using Bayes' Rule breaks P_{ij} down into probabilities that can be calculated (figure 5.1.1). $P(x_j \mid c=i)$ is the probability that x_j was generated by Gaussian c=i, it is calculated by plugging x_j and the values for Gaussian c=i into the multi-dimensional Gaussian function in figure 5.0.1. The component weight (ω_i) is the percentage of the data that the distribution covers. It is updated in the M-step. The last element necessary to complete

$$P_i = \sum_j P_{ij}$$

Figure 5.1.2

$$\sum_j x_j \cdot \frac{P_{ij}}{P_i} \rightarrow \mu_i$$

$$\sum_j (x_j - \mu_i) \cdot (x_j - \mu_i)^{\mathsf{T}} \cdot \frac{P_{ij}}{P_i} \rightarrow \varepsilon_i$$

$$\sum_j \frac{P_{ij}}{m} \rightarrow \omega_i$$

m = number of data points

Figure 5.2.1: M-Step

Bayes' Rule is P(x_j). The math shown in figure 5.1.1 provides a solvable answer for P(x_j). Now that all the information necessary to solve for P_{ij} is available, the E-step can be accomplished.

Every P_{ij} is calculated, and then the P_i values, the sum of the P_{ij} for each Gaussian (c=i), are calculated using the formula in figure 5.1.2.

5.2 M-Step

In the M-step the new mean, covariance and component weights are computed. Figure 5.2.1 breaks down the M-step, which is explained in the following paragraphs.

The component weight (ω_i) is unique for each Gaussian. It is the probability that the Gaussian was generated by the data. For example, if the Gaussian contains 25 data points out of 100, then $\omega = .25$. The component weight is equal to the P_i value for each Gaussian divided by m (the total number of data points). Because the data is Gaussian the sum of all P_i must equal one.

In order to compute the new mean for each Gaussian each data point (x_j, which is a d x 1 vector where d is the number of dimensions of the problem) is multiplied by the ratio P_{ij} / P_i, as calculated in the E-step. The P_{ij} / P_i ratio determines how likely x_j is included in the ith Gaussian. The new mean (μ_i) is calculated by summing P_{ij} / P_i * x_j for all j.

The new covariance matrix (ϵ_i) is calculated in the same manner as μ_i. The same ratio (P_{ij} / P_i) is used. However, because ϵ_i is a d x d matrix, additional calculations must been performed to create a d x d matrix out if a d x 1 matrix. Furthermore, ϵ_i must be symmetrical along the diagonal. In order to have both of these properties hold true, the covariance is created by multiplying (x_j - μ_i) by $(x_j - \mu_i)^{\mathsf{T}}$. The $^{\mathsf{T}}$ means that the matrix is transposed. So in the M-step, transposing the second d x 1 matrix creates a product that is a d x d matrix (d x 1 matrix * 1 x d matrix = d x d matrix) and is symmetrical along the diagonal. The subtraction (x_j - μ_i) is done to keep the values of the covariance matrix from becoming so large that they cover every data point.

These new mean, covariance, and component values are then passed back to the E-step and the process is begun anew. After about ten iterations of the EM algorithm the mean, covariance, and component values become relatively constant between iterations and are almost identical to the initial distributions. This means that our implementation was correct and it is time to begin implementing the data.

5.3 Implementation on Problem Data

When we apply the EM algorithm to our segmentation problem, we are expecting certain results. We believe that the mean and covariance matrices will create Gaussian distributions that model the data that performs a specific action. This way, by applying the Gaussian function in figure 5.0.1 with the newly derived mean and covariance matrices we will be able to determine when each action starts and stops. Inserting the start and stop points will segment the data into the actions that make up the planning units referred to in section 2.

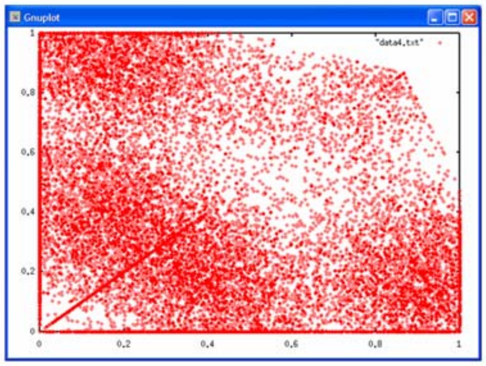
6 Methods

To implement the EM algorithm we wrote a C++ program. We used a header file downloaded from www.techsoftpl.com that performs the necessary matrix calculations. The file is written so that the user inputs the number of Gaussian distributions contained within the data along with the number of dimensions that the problem data is in. For example, if the data is in five columns then the data is in five dimensions. The program reads in the data and prints out the values for each Gaussian distribution after every iteration of the algorithm. The algorithm performs 10 iterations and then finishes, printing the final distributions to the screen.

7 Results

The first set of data analyzed contained four dimensions. Two of the dimensions represented the motor movements of wheels, their values ranged from zero to one. The other two dimensions represented the robot's arm movements. Their values were either a one or a zero. While running the algorithm on the data, the generated covariance matrix shrank to zero and the probabilities of the data became infinite. This occurred because the arms data contained only three possible values, (0,0), (0,1), or (1,0). This essentially created three separate points instead of three separate distributions. If a Gaussian zeroes in on a single point its variance shrinks to zero. This happens because the variance is designed to form a net around all of the points in the distribution; this net dictates which points belong to the distribution and which ones do not. A distribution that contains only one point needs a very small variance. When the variance gets this small, the probability of the point within the Gaussian becomes infinite. This crashes the algorithm.

After realizing this, we decided to remove the data containing the arm movements. Now the data is only two dimensions, a plot of it is shown in figure 7.0.1. Running the EM algorithm on this new set of data yielded predictable answers. There are obviously three major distributions amongst the data, and the EM algorithm could find them. However, EM did not always return the same set of Gaussian distributions. For example, the algorithm would sometimes split the data in the lower left corner of the graph amongst two distributions, or one Gaussian would cover the left side of the graph



7.0.1: A 2-Dimensional plot of the test Data

and the other two would cover the right side of the graph. In certain conditions, all of the Gaussians would center themselves in the middle of the graph. The above situations were a result of where the means of the distributions were initialized. Because the data on the graph was so dense, the distributions often changed their mean values only slightly and expanded their covariance matrices to cover as many points as possible. This could be avoided if the data was less dense.

Also, after graphing the two dimensional motor data, there appears to be a linear function in the lower left corner of the graph. To accurately model this data, a Kalman filter should be implemented.

8 Conclusion

After analyzing the available data, we have concluded that further testing is needed. The data could be segmented in the two dimensional implementation. However, the data was unable to be segmented in four dimensions. This was a result of having only a one or zero value for the arm movements. In order to fully conclude if using the EM algorithm is a viable way to segment the low level actions of a demonstrating machine, more data should be tested. This other test data should have a greater diversity amongst itself, i.e. no one or zero components. In conclusion, the EM algorithm should be tested more thoroughly before concluding that it is unable to segment the imitation learning data.

References

[1] F. L. Crabbe and R. Hwa. Robot imitation learning of high-level planning information. USNA Computer Science Department Technical Report USNA-CS-TR-2005-03.

[2] T Inamura, I. Toshima, and Y. Nakamura. Acquisition and embodiment of motion elements in close mimesis loop. In *Proceedings of the IEEE International Conference of Robotics and Automation*, pages 1539-1544, 2002.

[3] M. J. Mataric. Learning to behave socially. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, 1994.

[4] S. Russell and P. Norvig. Artificial intelligence a modern approach. Second edition. Pearson Education, Singapore 2003.